[Android 高效开发]

[Android 开发规范、开发中常见的技术问题和误区。参考其他各种资料（包括 CSDN.com、CNBLOG.com、Google 开发文档等），旨在提升开发效率（速度和质量）。]

[成都市天府软件园]
[tsotumu@163.com]
2019 年 1 月 16 日

Android 高效开发

版本说明

版本号	作者	更新日期	备注
1.0	tsotumu@163.com	2018 年 7 月 18 日星期三	添加 Android 编程常识

内容说明：

此文档涉及 Android 开发规范、开发中常见的技术问题和误区、部分技术关键点和难点以及常用库的源码解析。参考其他各种资料（包括 csdn.com、github.com、jianshu.com 以及 Google 开发文档等），旨在提升开发效率（速度和质量）。此文档的重点在于学习 Android 系统源码的前沿编码风格、规范以及思想，以提升代码写作能力。目的在于写出易于维护的、简练且运行效率高的代码，相关代码参加 AndroidDevSet。

目录

第一章 顶层设计

如何架构一个应用的开发。（参考网址 1，参考网址 2，参考链接）

（一）进程

如果一个进程占用内存超过了这个内存限制，就会报 OOM 的问题，很多涉及到大图片的频繁操作或者需要读取一大段数据在内存中使用时，很容易报 OOM 的问题。为了彻底地解决应用内存的问题，Android 引入了多进程的概念。

它允许在同一个应用内，为了分担主进程的压力，将占用内存的某些页面单独开一个进程，比如 Flash、视频播放页面，频繁绘制的页面等。在编写 Android 应用程序时，我们一般将一些计算型的逻辑放在一个独立的进程来处理，这样主进程仍然可以流畅地响应界面事件，提高用户体验。Android 系统为我们提供了一个 Service 类，我们可以实现一个以 Service 为基类的服务子类，在里面实现自己的计算型逻辑，然后在主进程通过 startService 函数来启动这个服务。（参考链接。）启动页改为单独进程，会提高启动响应速度？多进程是解决响应卡顿的备选方案吗？多进程会不会增加电量消耗或内存消耗？Manifest 里面声明的 activity 如何做到跨进程调用？把 Activity 生命为 **multiprocess = true** 有何意义呢？其他组件设置为 multiprocess=true 有何用处呢？（参考链接）

Android 中的进程和 Linux 中的进程有何区别？Android 的进程概念是什么？是否 Android 的一个虚拟机就是一个进程？一个 App 的进程是否就是一个 Android 虚拟机？每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。而每一个 DVM 都是在 Linux 中的一个进程，所以说可以认为是同一个概念，那么一个 app 里面的每一个进程都是一个独立的虚拟机实例吗？

1．进程特点

Android 应用的进程都是从一个叫做 Zygote 的进程 fork 出来的。Zygote 进程在系统启动，并载入通用的 framework 的代码与资源之后开始启动。为了启动一个新的程序进程，系统会 fork Zygote 进程生成一个新的进程，然后在新的进程中加载并运行应用程序的代码。这就使得大多数的 RAM pages 被用来分配给 framework 的代码，同时促使 RAM 资源能够在应用的所有进程之间进行共享。（参考链接 1。参考链接 2。参考网址）。总结如下：

(1) 进程是系统资源和分配的基本单位，而线程是调度的基本单位；
(2) 每个进程都有自己独立的资源和内存空间；
(3) 其它进程不能任意访问当前进程的内存和资源；
(4) 系统给每个进程分配的内存会有限制。

2．进程间如何通信。

在 android SDK 中提供了 4 种用于跨进程通讯的方式。这 4 种方式正好对应于 android 系统中 4 种应用程序组件：Activity、Content Provider、Broadcast 和 Service。其中 Activity 可以跨进程调用其他应用程序的 Activity；Content Provider 可以跨进程访问其他应用程序中的数据（以 Cursor 对象形式返回），当然，也可以对其他应用程序的数据进行增、删、改操 作；Broadcast 可以向 android 系统中所有应用程序发送广播，而需要跨进程通讯的应用程序可以监听这些广播；Service 和 Content Provider 类似，也可以访问其他应用程序中的数据，但不同的是，Content Provider 返回的是 Cursor 对象，而 Service 返回的是 Java 对象，这种可以跨进程通讯的服务叫 AIDL 服务。总结如下：

(1) 使用 Messenger；
(2) AIDL；
(3) 待添加。

3．Linux 进程优先级

在 Linux 系统中，优先级低的进程更容易被 kll。划分优先级如下：

(1) Foreground Process（焦点所在进程）
(2) Visible Process
(3) Service Process
(4) Background Process
(5) Empty Process

4．关于架构

应用启动的第一步是创建进程，那么如何设计这个进程呢？如何架构整个应用的开发呢？有如下思考点：

(1) 什么情况下，Service 需要运行在 Application 相同的进程？什么情况下，需要放在一个进程里面？
(2) 后台周期性运行的服务（比如 IntentService, Service, JobService, JobScheduler 等）需要运行在一个单独的进程吗？（参考网址：网址 1.）

（二）关于 Application

1．疑问

如何看待 Application，总结问题如下：

(1) Android 程序的入口是 ActivityThread.main(String[] args)。

(2) onCreate 在主线程执行吗？执行时机是什么？

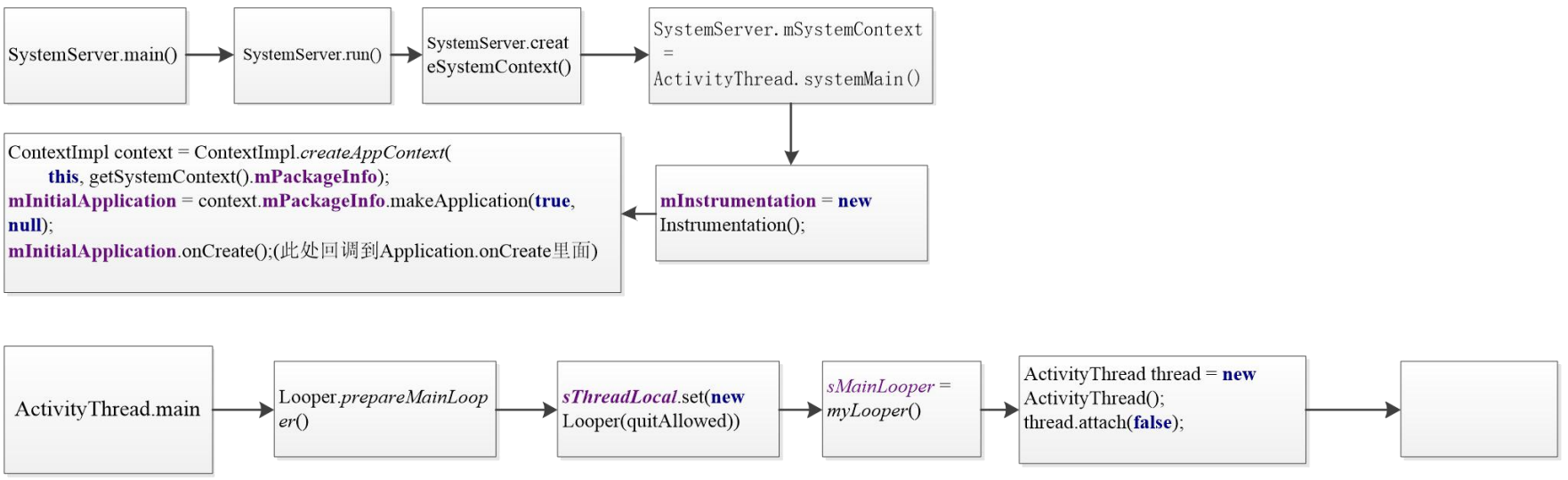
(3) 四大组件的 onCreate 优先于 Application 执行。

(4) 爱的色放多进程和 Application 的关系。有哪些注意事项；Application 里面哪些需要做进程区分的？进程之间共享数据是怎么样的？多次初始化。参考链接 1，消除卡顿，参考网址 1，参考网址 2，参考网址 3。

（三）从点击 App 图标说起

1．App 启动过程讲解

应用程序的启动过程实际上就是应用程序中的默认 Activity 的启动过程。首先是系统加载程序。（参考连接 1，参考链接，参考链接， 参考链接。）。然后运行 ActivityThread.main(String[] args)。从点击桌面图标开始，Android 的执行流程图如下所示：



（四）Binder 通信机制

在 Linux 中的 RPC 方式有管道，消息队列，共享内存等，消息队列和管道采用存储-转发方式，即数据先从发送方缓存区拷贝到内核开辟的缓存区中，然后再从内核缓存区拷贝到接收方缓存区，这样就有两次拷贝过程。共享内存不需要拷贝，但控制复杂，难以使用。Binder 是个折中的方案，只需要拷贝一次就行了。其次 Binder 的安全性比较好，好在哪里，在下还不是很清楚，基于安全性和传输的效率考虑，选择了 Binder。Binder 的英文意思是粘结剂，Binder 对象是一个可以跨进程引用的对象，它的实体位于一个进程中，这个进程一般是 Server 端，该对象提供了一套方法用以实现对服务的请求，而它的引用却遍布于系统的各个进程（Client 端）之中，这样 Client 通过 Binder 的引用访问 Server，所以说，Binder 就像胶水一样。

(1) 工作在 Linux 层面。属于驱动，运行在内核态。

(2) 分为三部分：服务端接口、Binder 驱动、客户端接口。参考链接。

（五）通用框架设计

框架设计包括：网络库、io 库、媒体播放、图片加载、模块间的消息库、多进程设计、跨进程通信、UI 库（符合 Material Design 风格）。

（六）线程

线程挂起，休眠，释放资源相关，唤醒，线程同步，数据传递。多线程（关于 AsyncTask 缺陷引发的思考）。

1．关于 Handler

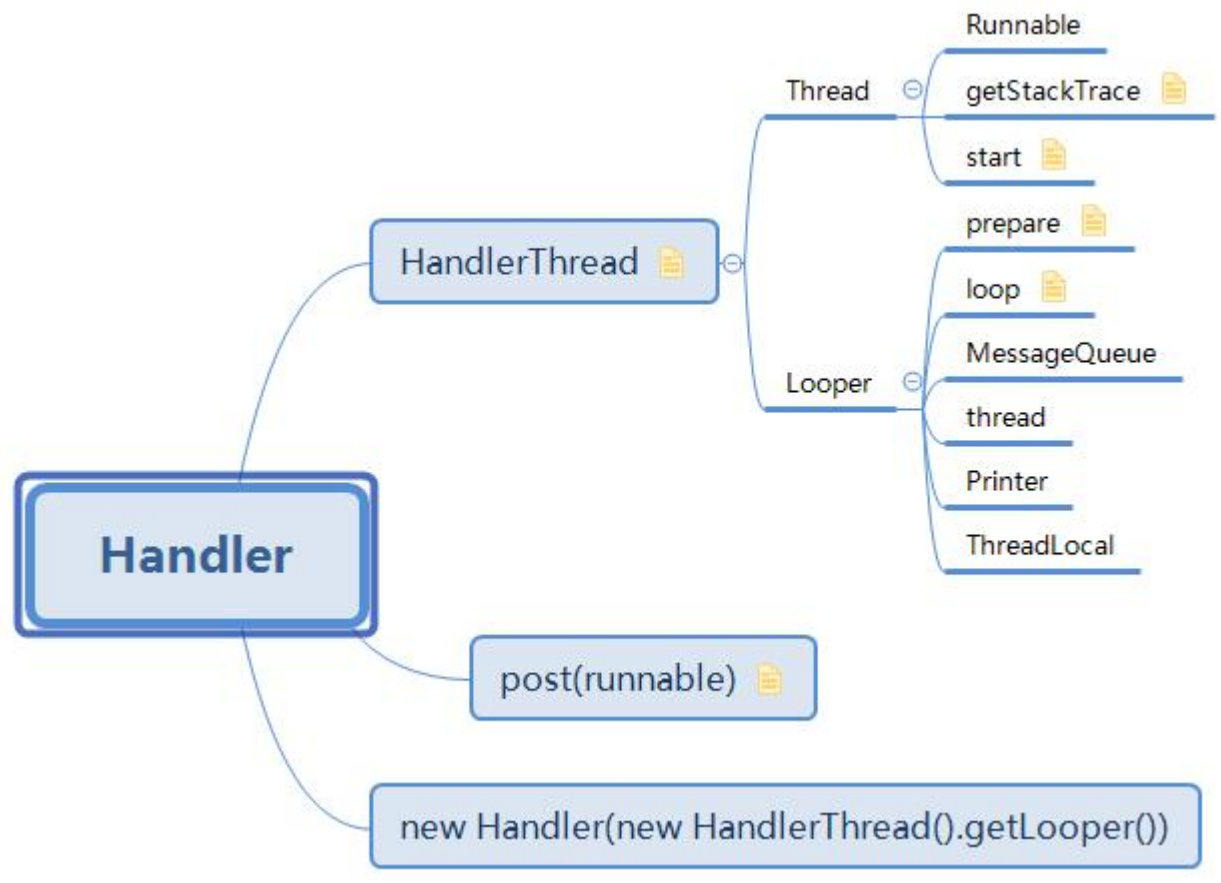
在 UI 线程新建一个 HandlerMessage 有何意义？参考网址。handler 发消息给子线程，looper 怎么启动？handler 实现机制（很多细节需要关注：如线程如何建立和退出消息循环等等）？如何实现主线程和异步线程的切换呢？（HandlerThread 中用到了 wait 和 notifyAll）

Android 应用程序的主线程在进入消息循环过程前，会在内部创建一个 Linux 管道（Pipe），这个管道的作用是使得 Android 应用程序主线程在消息队列为空时可以进入空闲等待状态，这样 CPU 并不会消耗太多资源在当前这个线程，并且使得当应用程序的消息队列有消息需要处理时唤醒应用程序的主线程。这里就涉及到 Linux pipe/epoll 机制，简单说就是在主线程的 MessageQueue 没有消息时，便阻塞在 loop 的 queue.next()中的 nativePollOnce()方法里，详情见 Android 消息机制 1-Handler(Java 层)，此时主线程会释放 CPU 资源进入休眠状态，直到下个消息到达或者有事务发生，通过往 pipe 管道写端写入数据来唤醒主线程工作。这里采用的 epoll 机制，是一种 IO 多路复用机制，可以同时监

控多个描述符，当某个描述符就绪(读或写就绪)，则立刻通知相应程序进行读或写操作，本质同步 I/O，即读写是阻塞的。 所以说，主线程大多数时候都是处于休眠状态，并不会消耗大量 CPU 资源，如下图箭头处所示（参考链接）：

```
/**
 * Run the message queue in this thread. Be sure to call
 * {@link #quit()} to end the loop.
 */
public static void loop() {
    final Looper me = myLooper();
    if (me == null) {
        throw new RuntimeException("No Looper; Looper.prepare() wasn't called on this thread.");
    }
    final MessageQueue queue = me.mQueue;

    // Make sure the identity of this thread is that of the local process,
    // and keep track of what that identity token actually is.
    Binder.clearCallingIdentity();
    final long ident = Binder.clearCallingIdentity();

    for (;;) {
        Message msg = queue.next(); // might block
        if (msg == null) {
            // No message indicates that the message queue is quitting.
            return;
        }

        // This must be in a local variable, in case a UI event sets the logger
        Printer logging = me.mLogging;
        if (logging != null) {
            logging.println(">>>>> Dispatching to " + msg.target + " " +
                    msg.callback + ": " + msg.what);
        }

        msg.target.dispatchMessage(msg);

        if (logging != null) {
            logging.println("<<<<< Finished to " + msg.target + " " + msg.callback);
        }
```

Handler 持有一个线程的 Looper 对象，这个 Looper 对象执行一个无线循环，持续从消息队列 MessgeQueue 取消息，如果有消息，此线程就阻塞在 next 函数，Android 中线程结构如下：



2．关于 MessageQueue.IdleHandler

在 looper 里面的 message 暂时处理完了，这个时候会回调这个接口，返回 false，那么就会移除它，返回 true 就会在下次 message 处理完了的时候继续回调。

使用场景：Android 框架中有一个使用它的应用场景。当从一个 Activity 启动另一个 Activity 时，Ams 要先调用当前 Activity 的 onPause()，然后再启动目标 Activity，目标 Activity 启动后，要把原 Activity 完全覆盖掉，按照 Activity 的生命周期管理，原 Activity 要

调用 onStop()，但它是在什么时候调用的呢，就是在 IdleHandler 中调用的。为什么这样做呢，因为新 Activity 的显示非常重要，要保证它的优先处理，旧 Activity 要销毁了，已经不显示 UI 界面了，那就等到 UI 线程处于空闲期再处理。我们知道 Android 应用运行启动之后，一般要进行全局初始化、资源加载、UI 显示等一系列的操作，显然此时会出现 CPU 使用一个高峰期，如果这些任务不加区分，一股脑的全部运行起来，势必影响到 UI 的显示。因此，可以对此进行流程优化，全力把保障 UI 线程的运行，把初始化、资源加载分为两部分，一部分是 UI 界面显示必需的，另一部分和初始化界面不相关的，在应用启动阶段只运行和 UI 相关的部分，其余部分等到高峰期过去之后再运行。当 UI 线程处理完 MessageQueue 中的所有 message 之后，进入了 idle 状态，此时 UI 的展现已经尘埃落定，正在等待用户的点击事件。虽然应用的其它普通任务可能也在运行，但是相对来说，没有 UI 功能那么重要，此时开始进行一些后台初始化的工作是非常恰当的，不过这些工作应该是启动线程异步运行的，以免占用 UI 线程。

想要在某个 activity 绘制完成去做一些事情，那这个时机是什么时候呢？有同学可能觉得 onResume()是一个合适的机会，不是可是这个 onResume() 真的是各种绘制都已经完成才回调的吗？用 View.postdelay()吗？那么 android 那些耗时的 measure, layout, draw 是在什么时候执行的呢？它们跟 onResume()又有何关系呢？将在 View 专题详细分析此问题。

3．AsyncTaskLoader

如果把 AsyncTask 比作一台烤面包机的话，那么 AsyncTaskLoader 就是操作烤面包机的面包师。AsyncTask 如同烤面包机接受命令完成面包的烤制任务，一旦任务完成它就停止了工作。然而 AsyncTaskLoader 如同面包师一样要根据顾客的需求来使用烤面包机。顾客会不停的光顾，那么面包师就会不停的使用烤面包机烤面包。后台执行任务也可以通过 IntentService，那么和异步线程有何不同呢？各自能解决什么问题呢？各自有何优势？（对于异步更新 UI 来说，IntentService 使用的是 Serivce+handler 或者广播的方式，而 AsyncTask 是 thread+handler 的方式。AsyncTask 比 IntentService 更加轻量级一点。Thread 的运行独立于 Activity，当 Activity 结束之后，如果没有结束 thread，那么这个 Activity 将不再持有该 thread 的引用。Service 不能在 onStart 方法中执行耗时操作，只能放在子线程中进行处理，当有新的 intent 请求过来都会线 onStartCommond 将其入队列，当第一个耗时操作结束后，就会处理下一个耗时操作(此时调用 onHandleIntent)，都执行完了自动执行 onDestory 销毁 IntengService 服务。）

4．AsyncTask

待添加。

第二章 四大组件

四大组件 activity、broadcastreceiver、ContentProvider、Service 都是运行在主线程，都会引起 ANR，所以做耗时操作，需要另起线程。

疑问：四大组件都是在同一进程的主线程运行，那是如何切换的呢？具体来说，四大组件的回调函数中哪些是在 UI 主线程执行的呢？主线程到底是什么？参考网址。怎么启动 service，service 和 activity 怎么进行数据交互？Android 系统为什么会设计 ContentProvider，进程共享和线程安全问题？

（一）Activity

Activity 的所有生命周期都在 ActivityThread 里面回调。实现 android 的 Activity 之间相互跳转需要用到 Intent，Intent 又分为显式 Intent 和隐式 Intent，没有在 Intent 里面明确指定 activity 的就是隐式，反之显式。intent-filter 里面有两个参数 action 和 category。

类别	参数	用途
Category	android.intent.category.LAUNCHER	如果要隐式启动的 activity 是 launcher，
Category	android.intent.category.DEFAULT	
Category		
Action	“<package>.intent.action.<action>”	
	“android.intent.action.MAIN”	
Data		
Extras		

1．Android 主题

主题优先于 xml 背景执行。

2．关于回退栈

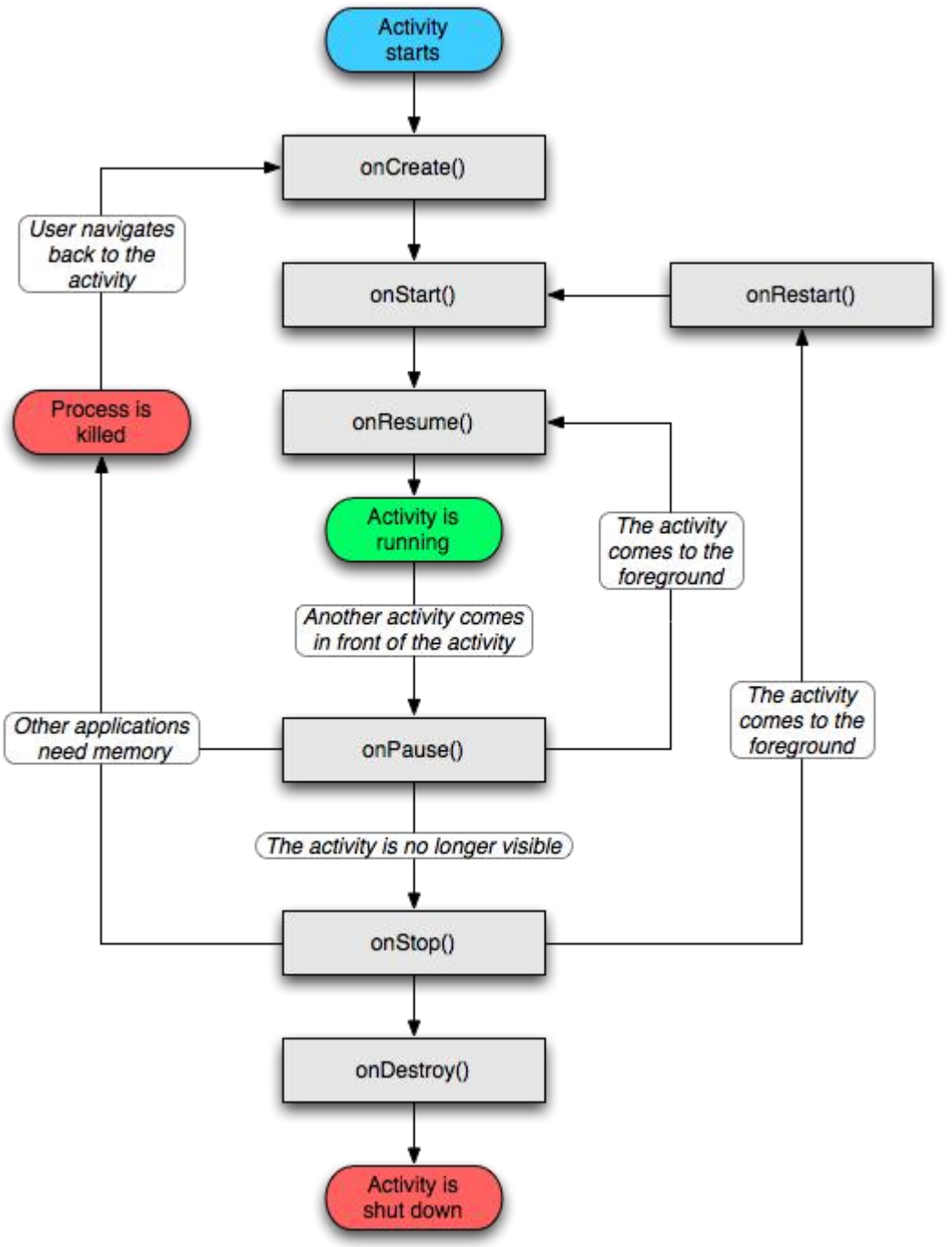
（1）阿斯蒂芬

（2）阿斯蒂芬

3．Activity 生命周期

Activity 生命周期执行顺序是：onCreate －> onContentChanged －> onStart －> onPostCreate －> onResume －> onPostResume －> onPause －> onStop －> onDestroy。onContentChanged()是 Activity 中的一个回调方法 当 Activity 的布局改动时，即 setContentView()或者

addContentView()方法执行完毕时就会调用该方法， 例如，Activity 中各种 View 的 findViewById()方法都可以放到该方法中。在 onResume 之前都没有执行 Measure 和 Layout 操作，这个是在后面的 performTraversals 中才执行的. 所以在这之前宽高都是 0，即调用 findViewById(R.id.iv_img).getHeight()为 0。PhoneWindow 与 Activity 是一对一的关系，PhoneWindow 中有很多大家比较熟悉的方法，比如 setContentView / addContentView 等，也有几个重要的内部类，比如：DecorView 。DecorView 是 PhoneWindow 的一个内部类。 DecorView 是我们当前 Activity 的最下面的布局。参考网址链接，顺着 getWindow().getDecorView().post(new Runnable()可以理解 activity 的绘制流程。



（1）OnCreate

onCreate 顾名思义就是 Create，我们在前面看到 Activity 的 onCreate 函数做了很多初始化的操作，包括 PhoneWindow/ DecorView/ StartingView/ setContentView 等，但是 onCreate 只是初始化了这些对象。应用从桌面启动的时候，在主 Activity 还没有显示的时候，如果主题没有设置窗口的背景，那么我们就会看到白色（这个和手机的 Rom 也有关系），如果应用启动很慢，那么用户得看好一会白色。如果要避免这个，则可以在 Application 或者 Activity 的 Theme 中设置 WindowBackground，这样就可以避免白色（当然现在各种大厂都是 SplashActivity+广告我也是可以理解的）。

（2）onStart

（3）onResume

真正要设置为显示则在 Resume 的时候，不过这些对开发者是透明了，具体可以看 ActivityThread 的 handleResumeActivity 函数，handleResumeActivity 中除了调用 Activity 的 onResume 回调之外，还初始化了几个比较重要的类：ViewRootImpl / ThreadedRenderer。主要是 wm.addView(decor, l); 这句，将 decorView 与 WindowManagerImpl 联系起来，这句最终会调用到 WindowManagerGlobal 的 addView 函数。（参考链接）

（4）OnPause

（5）onStop

（6）onDestroy

调用时机是在准备销毁这个 activity 之前吗？是在 onDestroy 里面释放在此 activity 引用的对象吗？调用 finish 后，会执行什么？isFinish()在何时会返回 true。OnDestroy 和内存释放之间有何关系？如何正确的释放资源才能保证 activity 被及时回收？是不是处理内存资源释放需要在 finish()里面。Activity 调用 ondetroy 后，还会执行 onresume 吗？如何保证 activity 相关资源被彻底释放？onDestroy 和内存回收有何关系？和 finalize 有何关系？需要调用 setContentView(R.layout.empty_layout)吗？activity 被回收后，其内部引用的资源也会自动回收吗？

当 activity finish() 的时候（按返回键，回到桌面），则 activity 不会被调用 onDestroy() ，原因可能是 activity 对象还在被引用！这句话该怎么理解？

如果发现 activity 对象还在被引用，是否 onDestroy 不会被调用？

你可以在 onPause() 方法里面判断 isFinishing() ，正常调用 finish() 后 activity 的回调过程是 onPause、onStop、onDestroy ，倘若出现上面的情况，只到 onPause！但是 isFinishing() 标志还是为 true ！你可以释放资源了。

官方说明：在 activity 被销毁（destroyed 是什么意思？）之前，执行最终的清理操作。触发时机有两个，其一是调用 finish 的时候，其二是系统为了减少内存消耗而销毁 activity 的对象，isFinish()可以辨别两种情况。不要在 onDestory 里面执行保持数据工作，可以在 onPause 或 onSaveInstance 保存数据。(内存泄露是指在准备销毁之前，该释放的资源没有释放吗？）在 GC 执行的时候，发现已经回调了 onDestroy 后的 activity 任然其他 GC ROOT 持有，就发生了内存泄露。

finalize()是在 GC 发现当前对象没有被引用的时候才会调用，且调用时间不一定，所以不能依赖这个调用释放 limited 资源。所以如果发生内存泄露，那么 finalize()方法就不会执行。（参考链接）OnDestroy 的内存回收和 Finalize 的内存回收有何区别？

（二）BrocastReceiver

1．描述

BrocastReceiver 生命周期只有十秒左右，如果在 onReceive() 内做超过十秒内的事情，就会报 ANR(Application No Response) 程序无响应的错误信息。

(1) 如果需要完成一项比较耗时的工作 ，应该通过发送 Intent 给 Service, 由 Service 来完成，不能使用子线程来解决，因为 BroadcastReceiver 的生命周期很短，子线程可能还没有结束 BroadcastReceiver 就先结束了。

(2) BroadcastReceiver 一旦结束，此时 BroadcastReceiver 的所在进程很容易在系统需要内存时被优先杀死，因为它属于空进程 (没有任何活动组件的进程)。 如果它的宿主进程被杀死，那么正在工作的子线程也会被杀死，所以采用子线程来解决是不可靠的。

(3) 被注册的广播会持有这个 context 对象吗？广播是保存在系统哪个地方？有谁调用的？既然是在主线程，那么如何和 UI 线程其他任务切换的？

(4) 注册过程和调用过程参考网址。

2．生命周期

（三）ContentProvider

（四）Service

Service 是运行在主进程的主线程（和绘制 UI 同一个线程）中。这意味着如果在 Service 中执行耗时操作，需要另开线程，否则会造成 ANR 或者界面卡顿。要判断一个任务是否在主线程中执行，Android 提供了 Thread.currentThread().getId()来得到当前线程的 id。既然在一个线程中，Service 和 activity 是如何实现解耦的呢？

(1) 使用场景：

① 仅仅执行不需要界面的任务。

② 由于 Service 相对于 Activity 生命周期可以比较长（Activity 在切换到后台后，会被系统调用 Destroy 销毁掉），可以在 Service 里面开启并维护一个全局的线程任务（即不受 activity 生命周期的约束的任务）。

(2) BindService

服务是如何绑定的？不同进程直接如何保证绑定的服务的方法能够执行？是 Android 的什么机制保证了这一行为？

(3) RegisterService

(4) IntentService

① Service 需要主动调用 stopService()；而 IntentService 不需要，所有 Intent 处理完后，系统自动关闭 IntentService。

(5) 前台服务（工具栏），有界面的服务，可以避免后台服务长时间执行，被系统 kill 的风险。

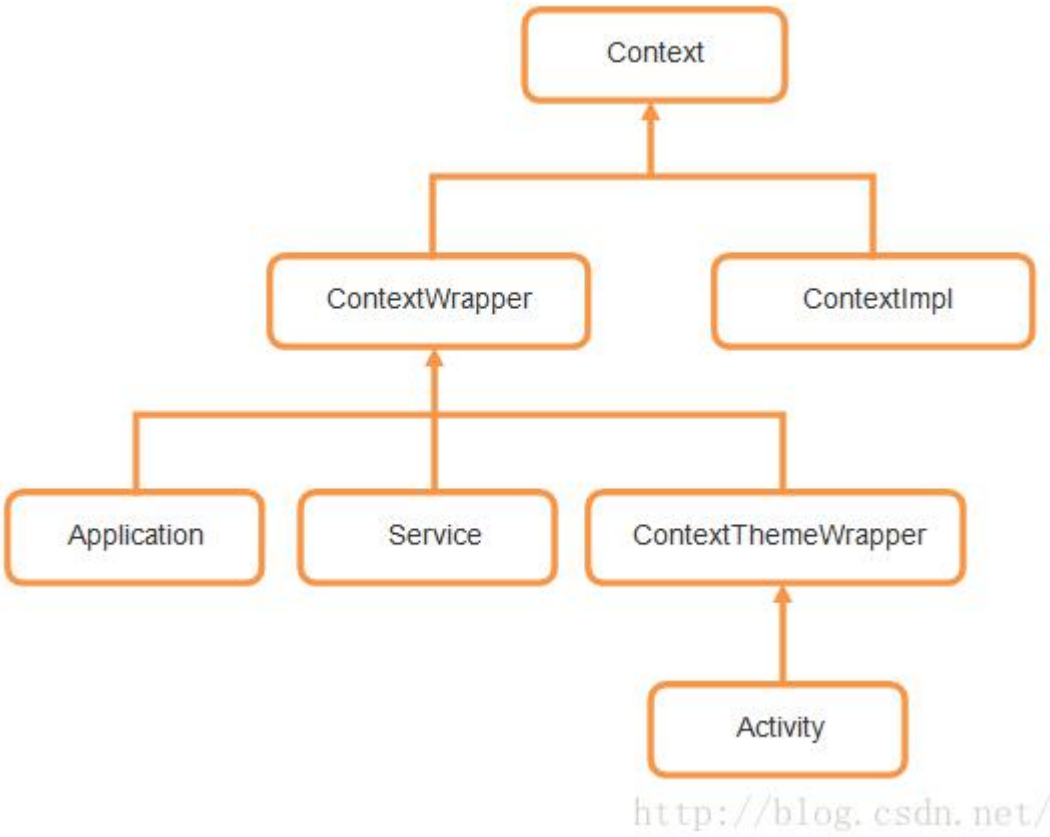
第三章 Context 解析

Abstract class Context 是维系 Android 程序中各组件正常工作的一个核心“功能类”。本身没有任何数据成员，只声明了一些方法。对 Context 的理解可以来说：Context 提供了一个应用的运行环境，在 Context 的大环境里，应用才可以访问资源，才能完成和其他组件、服务的交

互，Context 定义了一套基本的功能接口，我们可以理解为一套规范，而 Activity 和 Service 是实现这套规范的子类，这么说也许并不准确，因为这套规范实际是被 ContextImpl 类统一实现的，Activity 和 Service 只是继承并有选择性地重写了某些规范的实现。

（一）类组织结构

一个应用中的 Context 数量等于 Activity 的个数+Service 的个数+1（Application）。一个应用只存在一个 Application 对象，且通过 getApplication 和 getApplicationContext 得到的是同一个对象，两者的区别仅仅是返回类型不同。（参考链接）



(1) 子类

① ContextWrapper extends Context

作为 Context 的代理类，Context mBase 是其唯一的数据成员。其中 protected void attachBaseContext(Context base)方法提供了设置 mBase 的入口。

② ContextImpl extends Context

实现了 Context 所有方法。

③ ReceiverRestrictedContext extends ContextWrapper

a. 静态注册的 BroadcastReceiver 的 onReceive(Context context, Intent intent) 函数 conext 对象类型是 ReceiverRestrictedContext。

b. 不允许通过 registerReceiver(BroadcastReceiver receiver, IntentFilter filter)注册新的 BroadcastReceiver。但是可以传入 receiver = nul。

例如 Intent intent = registerReceiver(null, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));查询电量的时候可以通过上面这种方式返回一个 intent，从这个 intent 中能够拿到我们想要要的信息，而不是注册一个 BroadcastReceiver 不停回调 onReceiver 方法。

c. 屏蔽了 bindService，为什么？

```
@Override
public boolean bindService(Intent service, ServiceConnection conn, int flags) {
    throw new ReceiverCallNotAllowedException(
            "BroadcastReceiver components are not allowed to bind to services");
}
```

④ Activity extends ContextThemeWrapper extends ContextWrapper

⑤ Service extends ContextWrapper

⑥ Application extends ContextWrapper

（二）Context 提供的方法

Context 简单描述如下：

方法名	作用域	描述	使用场景	注意事项
startActivity				
startService				
sendBroadCast				
registerReceiver				
stopService				
unbindService				

从下表可以看出，Activity 所在 Context 没有限制，BroadcastReceiver 限制最多。只有在 Activity 才允许弹出对话框。所有的 Context 都

允许加载资源。

表 3.2.1

	Application	Activity	Service	ContentProvider	BroadcastReceiver
Show a dialog	N	Y	N	N	N
Start an activity	N(1)	Y	N(1)	N(1)	N(1)
Layout Inflation	N(2)	Y	N(2)	N(2)	N(2)
Start a Service	Y	Y	Y	Y	Y
Bind to a Service	Y	Y	Y	Y	N
Send a Broadcast	Y	Y	Y	Y	Y
Register RroadcastReceiver	Y	Y	Y	Y	N(3)
Load Resource Values	Y	Y	Y	Y	Y

表 3.2.2

其中 N(1)表示可以启动一个 activity，但是创建了一个新的回退栈。其中 N(2)是被允许的，但是只能用系统默认的主题，不允许使用应用中的自定义主题。其中 N(3)表示，在 Android4.2 以上，如果此 Receiver 是 null，是允许的。

（三）关键代码一览

Application 及其 BaseContext 的创建过程：

```
ContextImpl appContext = ContextImpl.createAppContext(mActivityThread, this);
app = mActivityThread.mInstrumentation.newApplication(
        cl, appClass, appContext);
```

（LoadedApk.java 里面的 makeApplication 函数。）

```
static ContextImpl createAppContext(ActivityThread mainThread, LoadedApk packageInfo) {
    if (packageInfo == null) throw new IllegalArgumentException("packageInfo");
    return new ContextImpl(null, mainThread,
            packageInfo, null, null, false, null, null);
}
```

（ContextIml.java 里面的 createAppContext 函数。）

```
static public Application newApplication(Class<?> clazz, Context context)
        throws InstantiationException, IllegalAccessException,
        ClassNotFoundException {
    Application app = (Application)clazz.newInstance();
    app.attach(context);
    return app;
}
```

（Instrumentation.java 里面的 newApplication 函数。）

```
/* package */ final void attach(Context context) {
    attachBaseContext(context);
    mLoadedApk = ContextImpl.getImpl(context).mPackageInfo;
}
```

（Application 的 attach 函数。）

```
protected void attachBaseContext(Context base) {
    if (mBase != null) {
        throw new IllegalStateException("Base context already set");
    }
    mBase = base;
}
```

（四）注意事项

(1) 静态对象持有对 Context 对象的引用而会造成的内存泄漏。

(2) 当 Application 的 Context 能搞定的情况下，并且生命周期长的对象，优先使用 Application 的 Context。

(3) 不要让生命周期长于 Activity 的对象持有到 Activity 的引用。

(4) Receiver 不是一个 Contex，为什么不能在 Receiver 里面的 intent 里面执行 startActivity.

(5) onReceive(Context context, Intent intent)返回的 Context 指的是此 BroadcastReceiver 对象的运行环境，因此对于注册方式不一样的 BroadcastReceiver，其回调函数 onReceiver 里面的 Context 是不一样的：静态注册的 BroadcastReceiver，返回的是 ReceiverRestrictedContext；应用内注册的，返回是 Application 对象；Activity 里面返回的是 Activity 对象；Service 里面注册的，返回的是 Service 对象；

(6) 既然 Context 只有一个实现类，并且 Activity,Service 以及 Application 都是继承自 ComtextWrapper,即都是 Context 的代理类，且其实现都是 ContextImpl。

(7) 三个的 Context 有何差别呢？在 Activity 里面如下图所示：

```
D/context_test: activity
                getApplication() = com.lm.powersecurity.app.ApplicationEx@a176c0d
                getApplicationContext() = com.lm.powersecurity.app.ApplicationEx@a176c0d
                getBaseContext() = android.app.ContextImpl@a4c9f02
                getApplication().getBaseContext() = android.app.ContextImpl@8348250
                getApplication().getApplicationContext() = com.lm.powersecurity.app.ApplicationEx@a176c0d
```

如上图所示：

① getApplication()，getApplicationContext()，getApplication.getApplicationContext()是用一个地址，即指向同一 Application 对象，因为 Application 间接继承自 Context；

② getBaseContext 是获取此 Activity 中的 ContextImpl 对象；

③ getApplication().getBaseContext 是获取的 Application 里面的 ComtextImpl 对象。

第四章 系统编译、链接、加载与库

（一）加载与运行

DexClassLoader 和 PathClassLoader 的区别。原理分析如下：

(1) Java 源程序（.java 文件）在经过 Java 编译器编译之后就被转换成 Java 字节代码（.class 文件）。类加载器负责读取 Java 字节代码，并转换成 java.lang.Class 类的一个实例。每个这样的实例用来表示一个 Java 类。通过此实例的 newInstance()方法就可以创建出该类的一个对象。实际的情况可能更加复杂，比如 Java 字节代码可能是通过工具动态生成的，也可能是通过网络下载的。

(2) Java.lang.ClassLoader 类的基本职责就是根据一个指定的类的名称，找到或者生成其对应的字节代码，然后从这些字节代码中定义出一个 Java 类，即 java.lang.Class 类的一个实例。除此之外，ClassLoader 还负责加载 Java 应用所需的资源，如图像文件和配置文件等。不过本文只讨论其加载类的功能。

（二）Android 系统编译与移植

1．系统编译

（参考链接）。

（三）库

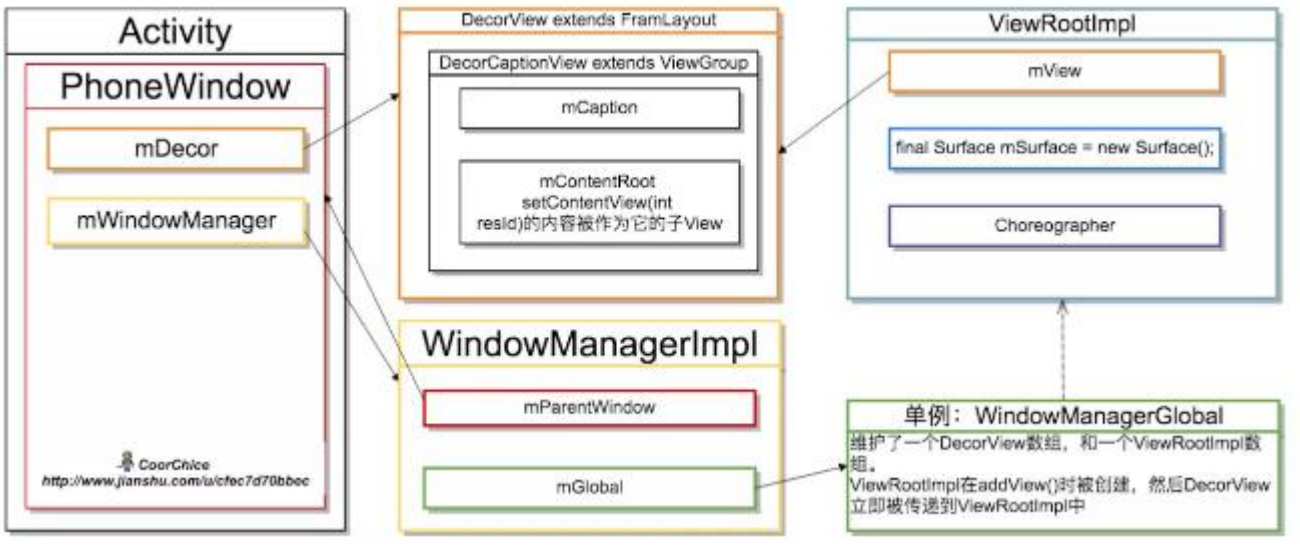
1．jni 开发

第五章 视图框架

Activity 和 Window，View 的关系，以及 DecorView 结构。自定义 View 里面 OnMeasure 参数的意义。

（一）基本框架

如下图所示：参考链接 1，



（二）View.Post

调用这个方法可以保证在 UI 线程中进行需要的操作，方便地进行异步通信。从本质上说，它还是依赖于以 Handler、LooperMessageQueue、

Message 为基础的异步消息处理机制。相对于新建 Handler 进行处理更加便捷。因为 attachInfo 中的 Handler 其实是由该 View 的 ViewRootImpl 提供的，所以 post 方法相当于把这个事件添加到了 UI 事件队列中。下面举一个常用的例子，比如在 onCreate 方法中获取某个 view 的宽高,而直接 View#getWidth 获取到的值是 0。要知道 View 显示到界面上需要经历 onMeasure、onLayout 和 onDraw 三个过程，而 View 的宽高是在 onLayout 阶段才能最终确定的，而在 Activity#onCreate 中并不能保证 View 已经执行到了 onLayout 方法，也就是说 Activity 的声明周期与 View 的绘制流程并不是一一绑定。

那为什么调用 post 方法就能起作用呢？首先 MessageQueue 是按顺序处理消息的，而在 setContentView()后队列中会包含一条询问是否完成布局的消息，而我们的任务通过 View#post 方法被添加到队列尾部，保证了在 layout 结束以后才执行。

因此对于“View.post 里获取到的 View 宽高是否准确”的问题，不能给出一个正确的答案，要解答这个问题，首先要理解透 performTraversals 和为什么 performTraversals 需要执行两次。（参考网址链接。参考链接。参考链接。）

（三）关于自定义 View

Af（参考链接。）封装 view 的时候怎么知道 view 的大小？

1．关于 inflate

View.Inflate(R.layout.view, null)会把 R.layout.view 的宽和高置为 wrap_content，其根布局的宽高将会失效。所以在使用 inflate 的时候需要注意两点：1）尽量用 View.Inflate(R.layout.view, parent)；2）在布局中，如果当前元素的高度能够确定，一定要用 wrap_content，不要用 match_parent，因为根元素的宽高可能会被重置为 wrap_content，导致 inflate 布局文件的时候，布局中的宽高失效。

（四）关于 View 的绘制

getMeasureHeight 何时能取到正确的值？

（五）关于事件传递

第六章 运行效率与优化

除了通过工具排查可以优化的点而外，还需要从用户体验和测试中找到可以优化的地方。比如卡顿的优化，就需要找到哪里卡顿，引起卡顿的原因，然后再找优化方案。参考网址 1.Android Studio 工具：Method trace 和 Systrace。

（一）用户交互流畅性

1) 数据库操作的优化。应用起的的时候调用 openDataBase 会导致应用启动卡顿。
2) 内存优化。
3) APP 为什么总是卡顿？参考链接 1，参考链接 2，参考网址 3.参考网址 4.参考网址 5.
4) 多线程，参考网址。
5) 频繁快速的向 Handler 提交任务，会有什么性能影响？

（二）启动优化

阿斯蒂芬

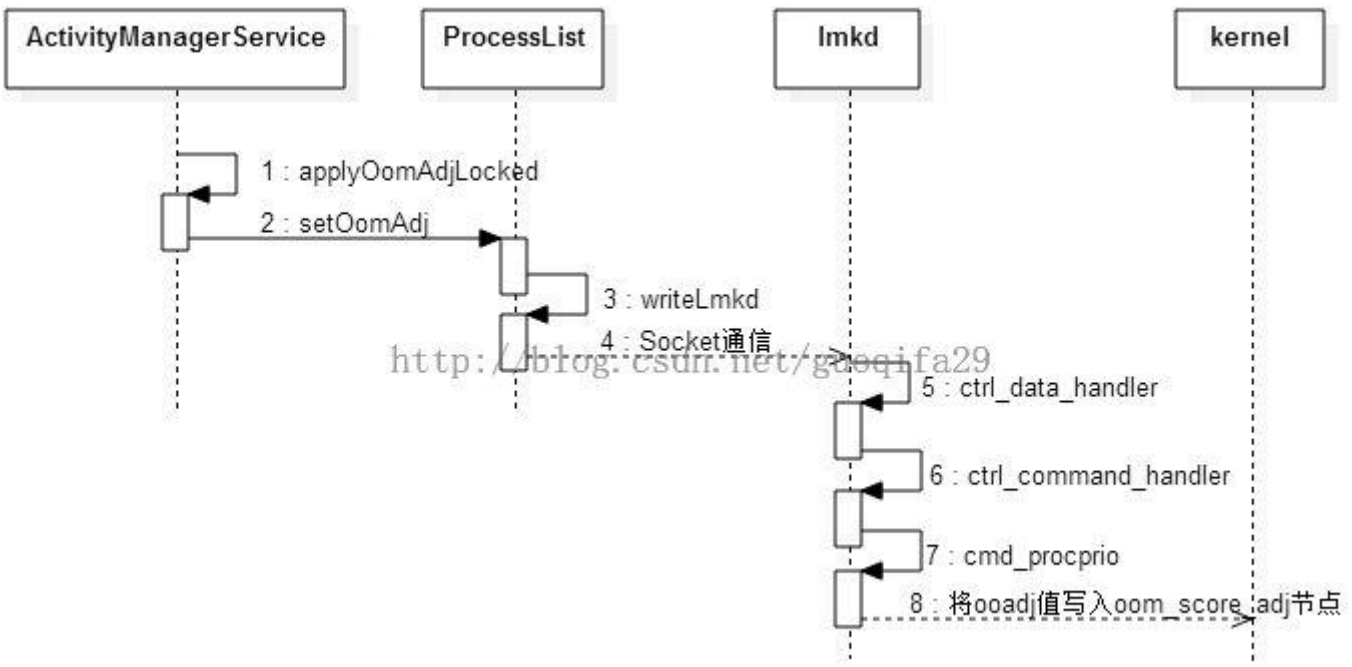
参考网址。参考网址 1。参考网址 3。把启动页放在单独进程可以解决启动卡顿吗？把 app 分拆成多个进程，可以达到优化启动卡顿的目的吗？一个 app 进程太多会不会占用更多的手机内存？分拆成多进程真的能解决单个进程内存不够的问题吗？

阿斯蒂芬

（三）内存优化

Android 进程回收主要涉及到两个组件：ActivityManagerService(AMS)和 lowmemoryKiller。进程对用户越不重要（Oomadj 值就越大），占用内存越大，进程就越容易被干掉。详细地说，当 app 状态发生改变时，比如退到后台时，AMS 会对 app 的进程计算出一个值，即 Oomadj（ams#computeOomAdjLocked），然后把这个值传给 linux 内核，lowmemorykiller 就可以拿到这个值了，lowmemorykiller 则就有了所有 app 进程的 Oomadj 值，即进程对用户的重要程度。当手机内存不足时，lowmemorykiller 就有了足够的信息决定干掉哪个进程了。

AMS 计算出一个危险的 Oomadj 值会调用 onTrimMemory 通知 app，此时 app 应该把不重要的内存释放掉，只要比友商 app 占用的内存小被 lowmemorykiller 干掉的概率就小。在收到 onTrimMemory 的时候，可以干掉其他应用，以保护自己。



1．内存相关建议

1) 安抚对于需要在静态内部类中使用非静态外部成员变量（如：Context、View ），可以在静态内部类中使用弱引用来引用外部类的变量来避免内存泄漏。
2) 阿斯蒂芬对于不再需要使用的对象，显示的将其赋值为 null，比如使用完 Bitmap 后先调用 recycle()，再赋为 null。
3) 如何正确加载图片：LRU、弱引用
4) 创建一个静态 Handler 内部类，然后对 Handler 持有的对象使用弱引用，这样在回收时也可以回收 Handler 持有的对象，这样虽然避免了 Activity 泄漏，不过 Looper 线程的消息队列中还是可能会有待处理的消息，所以我们在 Activity 的 Destroy 时或者 Stop 时应该移除消息队列中的消息，
5) 参考网址 1.参考网址 2.参考网址 3.参考网址 4.参考网址 5.参考网址 6.
6) 参考网址 1.参考网址 2.参考网址 3。参考网址 4.参考网址 5.参考网址 6.
7) 要回使用分析工具 MAT
8) 不要用枚举； float 类型的数据存取速度是 int 类型的一半，尽量优先采用 int 类型。

2．内存回收

内存分配参考网址。链接。Handler 会导致内存泄露（Activity 不能回收）的原因。所谓的 GC 根是指一个堆之外的对象。不可达对象可回收，内存管理分为新生代，老一代，持久代。BinderInternal：GcWatcher 有何作用？（参考链接， 参考链接）。

（1）关键字

和内存回收相关的调用有：System.gc()（告诉垃圾收集器打算进行垃圾收集，而垃圾收集器进不进行收集是不确定的），System.runFinalization()（强制调用已经失去引用的对象的 finalize 方法），Runtime.getRuntime.gc().

（2）View 相关。

① 无限循环的动画

无限循环的动画 setRepeatCount(INFINITE);如果没有 cancel 会导致对应的 view 得不到释放，进而会导致整个引用链上的对象不能被回收。代码分析如下：（动画参考链接、链接）

② View.post(runnable)

View.post 的执行时机,不同的时机得到的效果不一样,总体来说就是当 view 已经 attach 到 window 的时候，view 的 post 和 handler 的 post 是一致的，都是通过 handler 来进行消息分发。而但 view 未 attach 到 window 的时候,就是走的另外的消息机制，这种情况下就有可能产生内存泄露（参考链接）。

ThreadLocal 内部持有的实例是线程单利的，也就是不同的线程调用 sRunQueues.get()得到的不是同一个对象。ViewRootImpl 使用 ThreadLocal 来保存 RunQueue 实例，一般来说，ViewRootImpl#getRunQueue 都是在 UI 线程使用，所以 RunQueue 实例只有一个。UIThread 是应用程序启动的时候，新建的一个线程，生命周期与应用程序一致，也就是说 UI 线程对应的 RunQueue 实例是无法被回收的，但是无所谓，因为每次 ViewRootImpl#performTraversals 方法被调用时都会把 RunQueue 里的所有 Runnable 对象执行并清除。

当视图树尚未 attach 到 window 的时候，整个视图树是没有 Handler 的（其实自己可以 new，这里指的 handler 是 AttachInfo 里的），这时候用 RunQueue 来实现延迟执行 runnable 任务，并且 runnable 最终不会被加入到 MessageQueue 里，也不会被 Looper 执行，而是等到 ViewRootImpl 的下一个 performTraversals 时候，把 RunQueue 里的所有 runnable 都拿出来并执行，接着清空 RunQueue。由此可见 RunQueue 的作用类似于 MessageQueue，只不过，这里面的所有 runnable 最后的执行时机，是在下一个 performTraversals 到来的时候，MessageQueue 里的消息处理的则是下一次 loop 到来的时候。

3．图片存取

BitMap 对象回收，为什么一定要调用 recycle()。（参考链接）

（1）DiskLruCache

（参考链接。）

（2）Glide 库

Glide 有更加高效的内存管理。自动限制了图片在缓存和内存中的尺寸。相较而言，Picasso 缓存是全尺寸的，而 Glide 缓存的是和 Image View 相同的尺寸，即对不同的尺寸的 Image View 各缓存一份，即便展示的是相同内容。Glide 默认的 Bitmap 格式是 RGB_565，比 RGB_8888 格式的内存开销小一半。

（3）Facebook 图形库 Fresco

4、内存对齐

Zipalign 原理是什么？

（四）数据存取优化

频繁的写操作和频繁的读操作那个更加耗费性能？频繁的条件查找和频繁的 Select * From 哪个更加耗费性能？微信的聊天数据在本地都是加密处理的（防止 root 了被破解），设计一个类似的本地数据存储系统？

1．数据库查询优化

（1）关于查询

应尽量避免在 where 子句中使用!=或<>操作符。否则将引擎放弃使用索引而进行全表扫描；应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描；不论什么地方都不要使用 select * from t ，用详细的字段列表取代“*”，不要返回用不到的不论什么字段；操作大数据量时开启事务对 SQLite 进行优化；频繁的数据库查询操作，并且所查询的表通常为空，会不会影响到效率？

（2）关于存储

（3）关于删除

2．SP（SharedPreferences）优化

SP 为我们提供了轻量级存储能力，方便了少量数据的持久化。但是由于项目越来越庞大，SP 操作使用不当会导致 app 卡顿，乃至 ANR 问题。参考网址 1，参考网址 2。把 SP 放在多个文件里面有何好处（SP 的读写都会加锁，在并发的时候耗时会陡增，由于锁是针对 SP 实例对象，分拆成多个文件可以减少锁的阻塞）。有如下疑问：

① Apply 是在一个只有 1 个线程的线程池里面执行吗？
② handleStopActivity 是在哪个时机执行的（activity 的那个生命周期，Service 的哪个生命周期，Receiver 的哪个生命周期）？
③ SP 导致的 ANR 是因为主线程在等待把 SP 写入到文件系统的任务完成吗？
④ commit 和 apply 是哪个时候把数据写入磁盘的（commit 是同步写入磁盘，apply 是异步写入磁盘）。
⑤ 解决 ANR 也可考虑在异步线程调用 commit。
⑥ 为什么不支持跨进程共享？

3．Serializable 和 Parcelable 的区别

两者最大的区别在于存储媒介的不同，Serializable 使用 I/O 读写存储在硬盘上，而 Parcelable 是直接在内存中读写。很明显，内存的读写速度通常大于 IO 读写，所以在 Android 中传递数据优先选择 Parcelable。Serializable 会使用反射，序列化和反序列化过程需要大量 I/O 操

作，Parcelable 自已实现封送和解封（marshalled &unmarshalled）操作不需要用反射，数据也存放在 Native 内存中，效率要快很多。两个 Activity 之间传递对象还需要注意对象的大小，使用的 Binder 的缓冲区是有大小限制的（有些手机是 2 M），而一个进程默认有 16 个 Binder 线程，所以一个线程能占用的缓冲区就更小了（ 有人以前做过测试，大约一个线程可以占用 128 KB）。

（五）绘制优化

1．避免过度绘制

2．硬件加速

（六）电量优化

功耗分析，抓取 bugreport.用 historian 工具查看 wakelock 申请情况（参考链接）。

1．JobScheduler 实现后台任务优化

使用场景：1）应用具有您可以推迟的非面向用户的工作（定期数据库数据更新）；2）应用具有当插入设备时您希望优先执行的工作（充电时才希望执行的工作备份数据)；3）需要访问网络或 Wi-Fi 连接的任务(如向服务器拉取内置数据)；4） 希望作为一个批次定期运行的许多任务。JobScheduler 和 Android 6.0 出现的 Doze 都一样，总结来说就是限制应用频繁唤醒硬件，从而达到省电的效果。

2．WakeLock

当手机灭屏状态下保持一段时间后，系统会进入休眠，一些后台运行的任务就可能得不到正常执行，比如网络下载中断，后台播放音乐暂停等。WakeLock 正是为了解决这类问题，应用只要申请了 WakeLock，那么在释放 WakeLock 之前，系统不会进入休眠，即使在灭屏的状态下，应用要执行的任务依旧不会被系统打断。WakeLock 的使用需要谨慎处理，使用不当会让应用变成“电量杀手”。

（七）安装包优化

参考网址。混淆。

（八）Android 虚拟机

内存回收：参考网址。

（九）其他

1．保活

阿斯蒂芬（参考链接）。

2．Anr

(1) Asdf 当前发生 ANR 的应用进程被第一个添加进 firstPids 集合中，所以会第一个向 traces 文件中写入信息。反过来说，traces 文件中出现的第一个进程正常情况下就是发生 ANR 的那个进程。不过有时候会很不凑巧，发生 ANR 的进程还没有来得及输出 trace 信息，就由于某种原因退出了，所以偶尔会遇到 traces 文件中找不到发生 ANR 的进程信息的情况。
(2) 获取 anr 信息：setprop dalvik.vm.stack-trace-file /tmp/stack-traces.txt；adb dump /data/anr/traces.txt。
(3) 每次发生 ANR 时都会删除旧的 traces 文件，重新创建新文件。也就是说 Android 只保留最后一次发生 ANR 时的 traces 信息。
(4) 参考网址。

3．权限

官方网址。

4．Instrumentation

5．Adb shell

Asdf Adb shell 调试原理参考链接 1。

命令	描述
Adb shell dump meminfo “package name”	
Adb shell screenrecord /sdcard/aaaa.mp4 > adb pull /sdcard/aaa.mp4	录屏
Adb shell screencap /sdcard/aaa.png > adb pull /sdcard/aaa.png	获取当前 activity 栈
Adb shell dumpsys activity activities	
Adb shell pm clear “com.lm.powersecurity”	清楚应用数据
Adb shell pm list packages	列出手机应用
adb shell dumpsys gfxinfo "com.lm.powersecurity" - cmd trim 80	强制执行 onTrimMemory()回收应用内存

第七章 开发规范

（一）规范

(1) 尽量不要使用匿名对象。
(2) 要注意所有使用过的对象的生命周期，并且在界面需要销毁之前释放掉这些对象。
(3) 从整理优化 PS 代码的结果来看，在编码过程中，需要注意：1、内存及时释放；2、过度优化；3、线程锁导致 UI 卡顿；4、尽量不要用内部类；5、少操作 sp 和数据库查询。

（二）注解

(1) 使用注解改进代码检查
源代码阶段的注解如下表所示：（参考链接。）

注解	解释
@LayoutRes	定义资源类型变量
@interface	定义枚举常量

(2) 如何自定义注解
asdfa
(3) Butterknife 的实现
asdf

第八章 打包与发布

（一）自动打包

JenkinsAndroid 应用开发流程工具。

（二）关于 Gradle

1．关于 build.gradle

(1) Multidex，相关链接 1 链接 2 链接 3，参考链接 4，
(2) 相关参数配置解析如下：

参数	描述		其他
buildToolsVersion			
compileSdkVersion			导入 appcompat-v7 的时候，要求 buldToolsVersion 和 compileSdkVersion 版本为最新。
minSdkVersion			
targetSdkVersion			对权限有何影响？
multiDexEnabled			

buildConfigField			
defaultConfig			
repositories			
productFlavors			

（三）Proguard 语法

（四）升级

1．自动下载升级原理

升级工具 updatefun 原理。

第九章 逆向、二次打包与安全

（一）逆向

（二）编辑

（三）二次打包

（四）安全

1．加固原理

参考（参考链接， 参考链接）

2．Apktool 原理

3．混淆

（参考链接）

4．抓包

（参考链接）

第十章 Android 代码框架与设计模式

学习和分析开源项目的代码及其组织架构，以快速提升自代码写作和组织架构及思维的能力，此外还可以加深对 Android 开发的理解，快速提升 Android 开发能力，比如学习 Glide 源码，可以了解到 Android 处理大图片的相关知识，学习 ActiveAndroid 可以了解到 Sqlite 相关知识。学习和分析的对象包括 Android 系统源码、ActiveAndroid 数据库源码、GreenDao 源码、OkHttp 源码、LeakCanary 源码、Glide 源码。

设计模式相关（例如 Android 中哪里使用了观察者模式，单例模式相关）？程序员的编码水平基本和其所掌握的通用开源代码持平。查看源码是很紧急的事情，已经迫在眉睫了，要多看使用广泛的开源代码。（大公司的架构方案）

（一）享元模式

1．Handler 和 Looper 的实现。

Looper 是（参考链接。）

2．消息队列的实现

（二）ThreadPoolExecutor 的实现

线程池优化该怎么做？

（三）Android-job 的实现

（四）SharePreference 的实现

一个文件对应一个 SharedPreferences 对象。Put 和 get 操作都不涉及文件读写。只有调用了 commit 或 apply 才会将内存中的 map 写入文件。内部类 EditorImpl 封装了所有的读写操作。所以频繁地或多次 get 或 put 操作对性能是影响的，get 都是直接从 Map<String, Object> mMap 中返回对于的值，put 操作都是将值存储到 EditorImpl 对象的 Map<String, Object> mModified 里面。Commit 操作是；apply 操作是。Apply 和 commit 之前，都是先将 put 的值写入 map。关于加锁，读写会相互阻塞吗？那么频繁的 apply 对性能有何影响？何时执行 apply 合适？注意事项如下：

(1) 由于 SP 操作是保存在 map 对象里面，且第一次加载 xml 文件到内存会导致 UI 线程阻塞（加载完成之前，读取 SP 的线程会执行 wait），所以不适宜保存超大的 value。
(2) 每一次调用 edit()，会创建 EditorImpl 对象，多次调用是否会影响性能？
(3) 频繁调用 apply，会产生多个 runnable 在 QueuedWork 中，所有 runnable 都是在这个单线程中执行。activity 在执行到 handleStopActivity 会执行 QueuedWork.waitToFinish();所以需要一次性提交。
(4) 由于每次执行 apply 都是将整个 sp 的所有内容写入文件，比较耗时，所以不能连续 apply。
(5) 写入外存的时候会导致读阻塞吗？写入的时候会导致 UI 线程阻塞吗？
(6) 多线程中执行读写操作有何影响？
(7) SP 用写文件的方式实现了数据存储，相比数据库 SQLite 而言，两者的效率（存取速度，内存占用）如何？Android 下 SQLite 做了哪些优化？

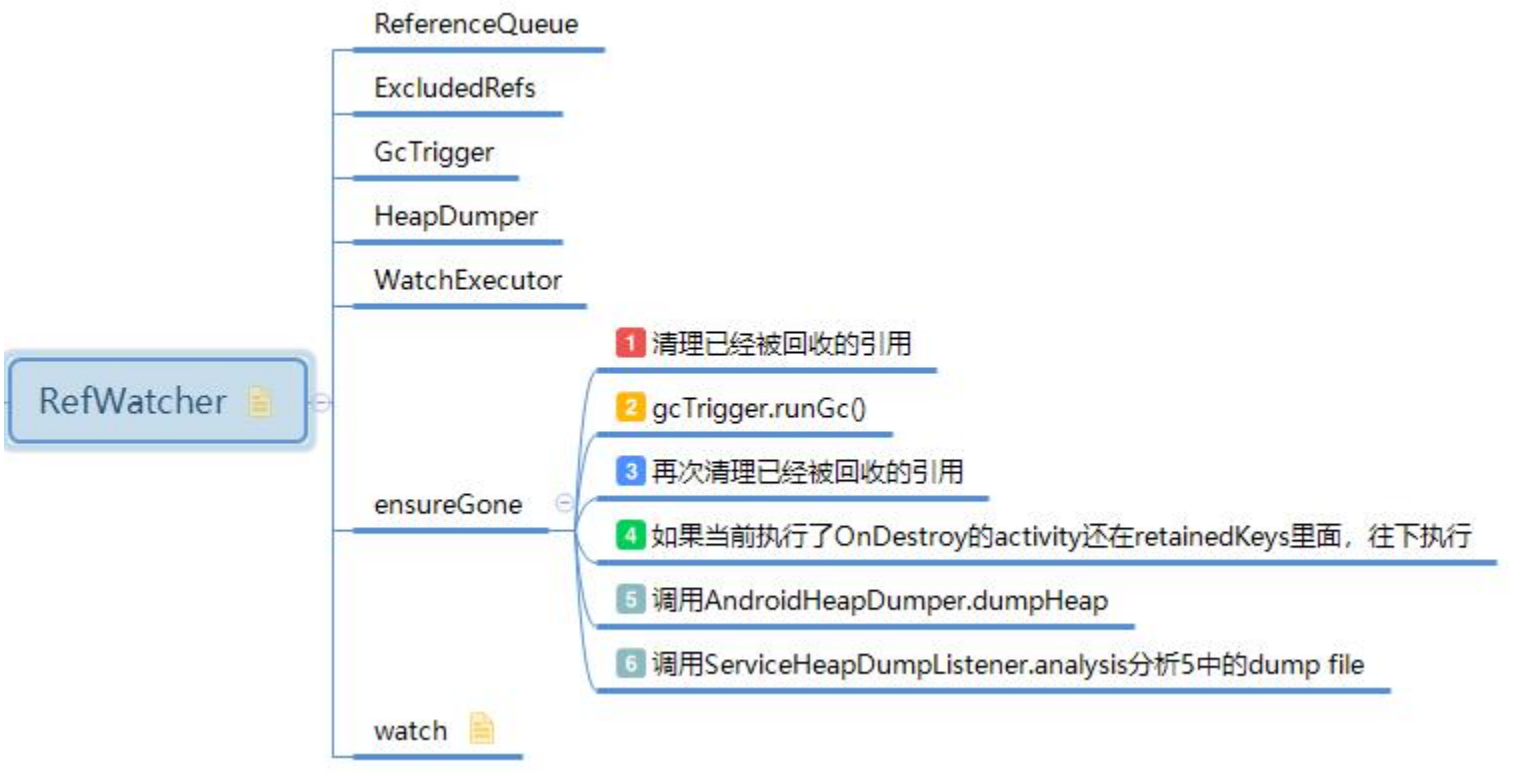
（五）LeakCanary、BlockCanary 的实现

1．LeakCanary 代码组织架构

分析其代码组织架构，以提升自己代码组织架构能力，目的在于一次构建出优秀的代码组织结构。

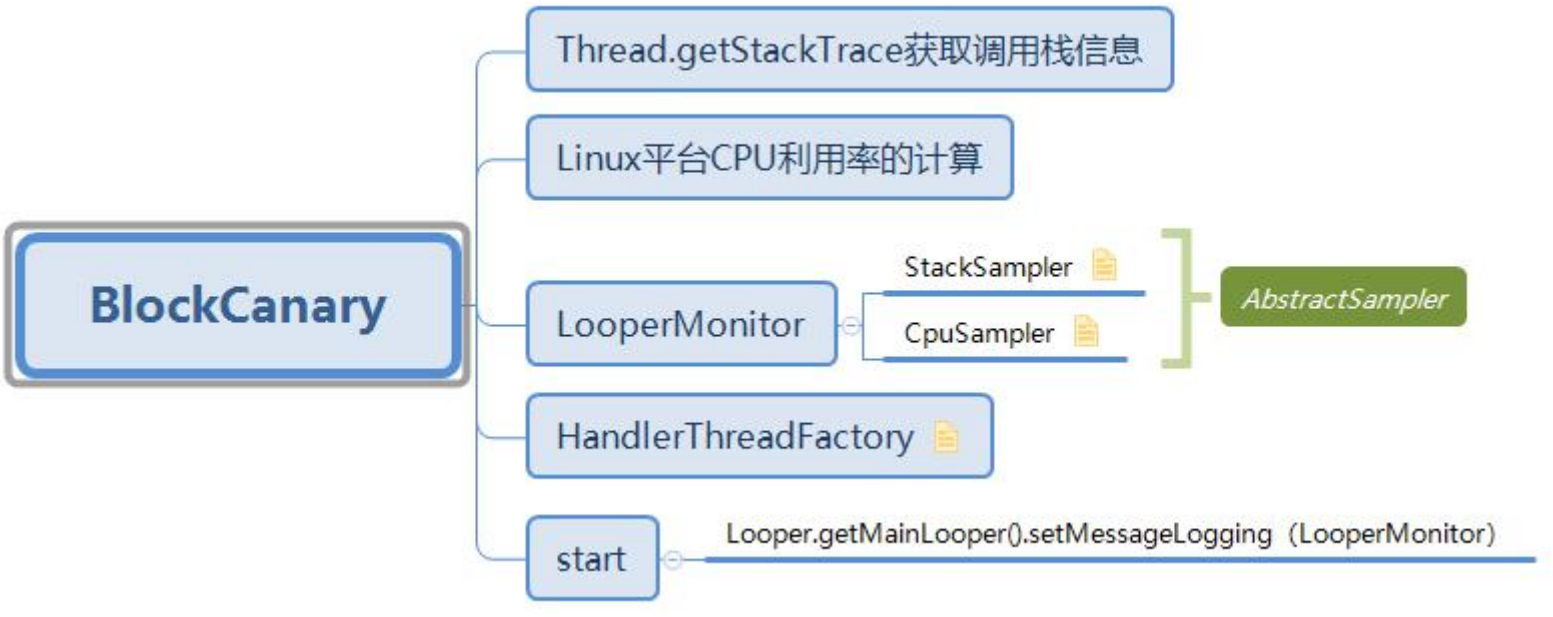
2．LeakCanary 实现原理

通过注册 ActivityLifecycleCallbacks 监听 activity 的 onDestroy 事件。在 onDestroy 回调的时候，判断这个 activity 是否可以被 GC 回收，如果没有被回收，则调用一次 gc，再次判断是否回收，如果此时没有被回收，则发生了内存泄露。OnDestroy 执行的时机是调用 finish()后或因为系统内存不足，而销毁当前 activity 的时候。关键是用到了 ReferenceQueue，在相关对象被回收的时候，此对象的指向着 Reference 对象会被加入到 ReferenceQueue，所以通过判断 ReferenceQueue 有无 Reference 对象，可以知道其所指向的对象是否被回收。



3．BlockCanary 实现原理

通过 Looper.getMainLooper().setMessageLogging（LooperMonitor）实现了对线程任务 msg.target.dispatchMessage(msg)执行时间的监控，其实现结构如下所示：



（六）Android 平台优化后的算法和容器

arraylist 和 linkedlist 的区别，以及应用场景？

各个容器如下：

容器	使用场景	优化的地方	缺陷
ArrayList			
HashSet			
SparseArray			
LruCache			

（七）ThreadPoolExcutor 的实现

一次抛出大量的 Runnable 执行，对性能有何影响？如何理解开启单个线程执行一个任务 QueuedWork. singleThreadExecutor(). execute (writeToDisk Runnable);。

（八）OKHppt、Volley 与网络优化

网络请求缓存处理，okhttp 如何处理网络缓存的？https 相关，如何验证证书的合法性，https 中哪里用了对称加密，哪里用了非对称加密，对加密算法（如 RSA）等是否有了解？多线程断点续传原理？

（九）Glide、Fresco 与图片优化

图片加载库相关，bitmap 如何处理大图，如一张 30M 的大图，如何预防 OOM？ listview 图片加载错乱的原理和解决方案？

（十）ActiveAndroid 源码解析

封装了数据库操作逻辑，实现了 gradle 插件、实现了数据库自动升级逻辑。

1．组织架构

Asdf

2．编码风格

（1）所有的 interface 内部都包含了一个默认实现的对象，此对象隐含为 public final。

（2）一个单独的文件代码行数不超过 200。

3．Gradle 插件

（十一）Greendao 代码自动生成插件与结构设计

（十二）阿里基础框架 Atlas 结构设计与技术

Atlas 是伴随着手机淘宝的不断发展而衍生出来的一个运行于 Android 系统上的一个容器化框架，我们也叫动态组件化(Dynamic Bundle)框架。它主要提供了解耦化、组件化、动态性的支持。覆盖了工程师的工程编码期、Apk 运行期以及后续运维期的各种问题。与插件化框架不同的是，Atlas 是一个组件框架，Atlas 不是一个多进程的框架，他主要完成的就是在运行环境中按需地去完成各个 bundle 的安装，加载类和资源。

1．技术原理

（网址）

2．组件化技术

第十一章 模块化编程

阿斯蒂芬

（一）组件化

参考马克宅的博客（链接）。

（二）热修复/插件化

1．底层替换方案

阿道夫

2．类加载方案

类加载方案基于 Dex 分包方案。1）随着应用功能越来越复杂，代码量不断地增大，引入的库也越来越多，可能会在编译时出现 65536 异常，这说明应用中引用的方法数超过了最大数 65536 个。产生这一问题的原因就是系统的 65536 限制，65536 限制的主要原因是 DVM Bytecode 的限制，DVM 指令集的方法调用指令 invoke-kind 索引为 16bits，最多能引用 65535 个方法。2）在安装时可能会提示 INSTALL_FAILED_DEXOPT。产生的原因就是 LinearAlloc 限制，DVM 中的 LinearAlloc 是一个固定的缓存区，当方法数过多超出了缓存区的大小时会报错。

为了解决 65536 限制和 LinearAlloc 限制，从而产生了 Dex 分包方案。Dex 分包方案主要做的是在打包时将应用代码分成多个 Dex，将应用启动时必须用到的类和这些类的直接引用类放到主 Dex 中，其他代码放到次 Dex 中。当应用启动时先加载主 Dex，等到应用启动后再动态的加载次 Dex，从而缓解了主 Dex 的 65536 限制和 LinearAlloc 限制。Dex 分包方案主要有两种，分别是 Google 官方方案、Dex 自动拆包和动态加载方案。

3．Instant Run 方案

第十二章 交互优化

（一）MD 风格

MD 风格即使用类似 Android 提供的 MD 开发控件的 UI 风格，控件包括：

1．主题

参考网址。

2．状态栏一体化

实现状态栏一体化原理（参考博客链接）。 参考网址，网址 2，网址 3。

（二）V4 与 V7、V13 详解

1. 阿斯蒂芬

如果在低版本 Android 平台上开发一个应用程序，而应用程序又想使用高版本才拥有的功能，就需要使用 Support 库。参考网址，参考网址 2，参考网址 3，

(1) V13 只在开发平板机上使用。

2. AppCompatActivity

用于替代 ActionBarActivity，

第十三章 扩展

（一）技术规划

掌握技术遵循二八原则，首先抓住那占比很小的最核心的技术点，然后再扩展其他技术。即参考网址。一种依赖注入工具 Dagger2（参考链接。）。关于 Android 提高和技术路线成长的博客（链接）。Android 开发优秀博客列表（链接）。高级 Android 开发工程师技术成长规划。技术点如下所示：

阶段	技术点	难度	重要性
初级开发人员	如四大组件如何使用、如何创建 Service、如何进行布局、简单的自定义 View、动画等常见技术		
中级工程师	熟悉 AIDL，理解其工作原理，懂 transact 和 onTransact 的区别		
	从 Java 层大概理解 Binder 的工作原理，懂 Parcel 对象的使用		
	熟练掌握多进程的运行机制，懂 Messenger、Socket 等		
	事件分发：弹性滑动、滑动冲突等		
	玩转 View：View 的绘制原理、各种自定义 View		
	动画系列：熟悉 View 动画和属性动画的不同点，懂属性动画的工作原理		
	懂性能优化、熟悉 mat 等工具		
高级工程师			
资深工程师	了解 SystemServer 的启动过程		
	了解主线程的消息循环模型		
	了解 AMS 和 PMS 的工作原理		
	能够回答问题“一个应用存在多少个 Window？”		
	了解四大组件的大概工作流程		
	Activity 的启动模式以及异常情况下不同 Activity 的表现		
	Service 的 onBind 和 onReBind 的关联		
	onServiceDisconnected(ComponentName className)和 binderDied()的区别		
	AsyncTask 在不同版本上的表现细节		
	线程池的细节和参数配置		
	熟悉设计模式，有架构意识		